Two-Restricted One Context Unification is in Polynomial Time

Two-Restricted One Context Unification is in Polynomial Time*

Adrià Gascón[1], Manfred Schmidt-Schauß[2], and Ashish Tiwari[1]

1 **SRI International, Menlo Park, CA, USA**
adriagascon@gmail.com,tiwari@csl.sri.com
2 **Goethe-Universität, Frankfurt, Germany**
schauss@ki.informatik.uni-frankfurt.de

Abstract

One Context Unification (1CU) extends first-order unification by introducing a single context variable. This problem was recently shown to be in NP, but it is not known to be solvable in polynomial time. We show that the case of 1CU where the context variable occurs at most twice in the input (1CU2r) is solvable in polynomial time. Moreover, a polynomial representation of all solutions can be computed also in polynomial time. The 1CU2r problem is used as a subroutine in polynomial time algorithms for several more general classes of 1CU. Our algorithm can be seen as an extension of the usual rules of first-order unification and can be used to solve related problems in polynomial time, such as first-order unification of two terms that tolerates one clash, and several interesting classes of the general 1CU problem. All our results assume that the input terms are represented as Directed Acyclic Graphs.

1998 ACM Subject Classification F.3.2 Semantics of Programming Languages, F.4.1 Mathematical Logic, F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases context unification, first-order unification, deduction, type checking

Digital Object Identifier 10.4230/LIPIcs.CSL.2015.405

1 Introduction

The well-known first-order unification problem consists of solving equations between terms with leaf variables ranging over terms. Context unification (CU) extends first-order unification by introducing context variables of arity one standing for contexts. Hence, the CU equations may contain subterms of the form $F(s)$, where F is a context variable that may be instantiated by a context. For example, the equation $F(a) \doteq f(a, a)$ has two solutions, since applying either substitution $\{F \to f(a, \bullet)\}$ or $\{F \to f(\bullet, a)\}$, gives the trivial equation $f(a, a) \doteq f(a, a)$.

Context unification falls in between first-order unification, which is solvable in linear time [19], and higher-order unification, which is undecidable [10]. The best known upper bound for the complexity of context unification is PSPACE [12]. Moreover, several variants and specializations of context unification have also been studied [16, 20, 15, 14, 6, 17, 7]. This paper is concerned with a particular case of CU called the *One Context Unification (1CU)* problem. In 1CU, only one context variable occurs in the input terms, possibly with

* This work was sponsored, in part, by National Science Foundation under grants CCF-1423296 and CNS-1423298, and ONR under subaward 60106452-107484-C under prime grant N00014-12-1-0914. The views, opinions, and/or findings contained in this report are those of the authors and should not be interpreted as representing the official views or policies, either expressed or implied, of the funding agencies.

many occurrences. This problem was recently proved to be in NP [8], but whether it is NP-hard or solvable in polynomial time is still open. That also holds for the case where the input terms are represented with Singleton Tree Grammars [4], a compression mechanism for terms that is more general than Directed Acyclic Graphs (DAGs). The initial interest in one context unification comes from interprocedural program analysis [11, 5], where context variables are used to represent (the yet unknown) summaries of procedures. In particular, one context unification problems (over uninterpreted terms) arise when analyzing programs using an abstract domain consisting of (uninterpreted) terms.

An interesting simple variant of the 1CU problem is term unification up to one clash, or fault tolerant unification. For example, while $f(x, f(a, y))$ unifies with $f(f(a, b), x)$, changing an a into a b in one of these two terms makes them non-unifiable. The terms $f(x, f(a, y))$ and $f(f(b, b), x)$ are, however, almost unifiable: if we are willing to accept disagreement at one position, namely position 2.1, the rest of the terms unify. The position 2.1 is a *distinguishing position* of these two terms. Note that two first-order terms s, t almost unify if the restricted 1CU instance $F(z_1) = s, F(z_2) = t$ has a solution, where z_1, z_2 do not occur in s, t. Intuitively, the position of the hole of $F\sigma$ in the solution indicates the distinguishing position in fault tolerant unification. A particular application of fault tolerant unification is Milner-polymorphic type checking, where in case the type check/computation fails, a most probable reason for the fail can be computed and presented to the programmer.

In this paper, we consider the special case of the one context unification problem consisting of only two equations $F(r_1) \doteq s_1, F(r_2) \doteq s_2$, where r_1, r_2, s_1, s_2 are first-order terms without occurrences of F, which we call 2-restricted 1CU (1CU2r). We present a polynomial time algorithm for deciding 1CU2r. We also show that a representation of all solutions can be constructed in polynomial time, which can be used to effectively enumerate all unifiers of the problem. All our results hold also when the input terms are represented as DAGs.

The restriction to two equations (1CU2r) is motivated by two facts. First, the solution to the 1CU2r problem also solves the problem of finding (all) distinguishing positions for two given first-order terms. Second, in a recent paper [9], we presented polynomial time algorithms for several classes of 1CU problem (that have more than 2 equations, but have other restrictions on the terms), but the algorithm in [9] *assumes* that 1CU2r can be efficiently solved. The result in [9] can be interpreted as showing that the 1CU2r problem is the "hard" part in solving the general 1CU problem, and it possibly exhibits several of the intricacies that make 1CU challenging. Our results, combined with [9], may open a way to construct a polynomial time algorithm for the unrestricted one context unification problem.

2 Preliminaries

We assume knowledge of first order terms, substitutions, and first-order unification (see [3, 1, 2]), and use the following notation: $\mathcal{F}$ denotes a fixed finite ranked alphabet, $\mathcal{X}$ is a set containing first-order variables and exactly one context variable F, f, g denote function symbols, a, b denote constant symbols, x, y, z denote (first-order) variables, and p, q denote positions (sequences of positive integers) in terms. Our algorithm introduces fresh first-order variables from a set $\mathcal{Y}$, which we denote by y with possible subindexes. We denote $\mathcal{X} \cup \mathcal{Y}$ as $\mathcal{V}$. Hence, we will argue about terms in $\mathcal{T}(\mathcal{F}, \mathcal{X})$, $\mathcal{T}(\mathcal{F}, \mathcal{V})$ and $\mathcal{T}(\mathcal{F}, \mathcal{X}) \cup \mathcal{Y}$.

The set of positions of a term t, denoted by $\mathtt{pos}(t)$, is defined recursively as $\mathtt{pos}(f(t_1, \ldots, t_m)) = \{\lambda\} \cup \{i.p \mid i \in \{1, \ldots, m\} \wedge p \in \mathtt{pos}(t_i)\}$. The *length* of a position is denoted by $|p|$. The *subterm* of a term t at a position $p \in \mathtt{pos}(t)$, denoted $t|_p$, is defined recursively as $t|_\lambda = t$ and $f(t_1, \ldots, t_m)|_{i.p} = t_i|_p$. With $<$ we denote the prefix relation among positions

and with $\prec$ the subterm relation among terms. We say that two positions are *parallel* if they are incomparable by the prefix relation. We also define $\mathtt{topsymbol}(f(t_1,\ldots,t_n) = f$, and $\mathtt{topsymbol}(x) = x$, and $\mathtt{vars}(t) = \{x \in \mathcal{V} \mid t|_p = x\}$.

By $\mathtt{maxarity}$, we denote the maximum arity of the symbols in $\mathcal{F}$, and by $\mathtt{pos}(\mathcal{F})$, we denote the set of positions $\{\lambda\} \cup \{1,\ldots,\mathtt{maxarity}\}^+$. We also use contexts C, D, where $\mathtt{hp}(C)$ is the notation for the position of the hole, denoted $\bullet$. Analogously as for terms, we refer to contexts in $\mathcal{C}(\mathcal{F},\mathcal{X})$ and $\mathcal{C}(\mathcal{F},\mathcal{V})$. In the notation that mixes first-order terms, contexts and plugging terms into holes, we write $C[s]$ for the term where s is plugged into the hole of C, and CD for the context $C[D]$. Similarly, we write $F(s)$ for the application of the context variable F to the term s. Sometimes we will omit brackets, if this does not lead to confusion. Also, we denote by $t[s]_p$, the term obtained from t by replacing its subterm at position p by s. The *exponentiation* of a position p to a natural number n, denoted p^n, is the position recursively defined as $p^n = p.p^{n-|p|}$ if $n > |p| > 0$, and as $p^n = p_1$ if $n \leq |p|$, $p = p_1.p_2$, and $|p_1| = n$. Note that $p^0 = \lambda$. The *exponentiation* of a context C to a natural number n, denoted C^n, is defined analogously to p^n.

In this work, we deal with *equations on terms*, denoted by e with possible subindexes. Given an equation $e = (s \doteq t)$, we call the set $\{s,t\}$ the *topterms of* e, denoted $\mathtt{topterms}(e)$. Similarly, for a set Δ of equations, $\mathtt{topterms}(\Delta)$ denotes $\bigcup_{e\in\Delta}(\mathtt{topterms}(e))$. Similarly, $\mathtt{topvars}(e) = \mathtt{topterms}(e) \cap \mathcal{V}$. By $|\Delta|$ we denote the number of equations in Δ.

A *substitution*, denoted by σ, θ, η, is a total function $\sigma : \mathcal{V} \to \mathcal{T}(\mathcal{F},\mathcal{V}) \cup \mathcal{C}(\mathcal{F},\mathcal{V})$ such that $\alpha\sigma \in \mathcal{T}(\mathcal{F},\mathcal{V})$ if α is a first-order variable and $\alpha\sigma \in \mathcal{C}(\mathcal{F},\mathcal{V})$ if α is a context variable. Substitutions are extended, in the usual way, to be mappings from terms to terms and contexts to contexts. We also extend the notion of application of a substitution to equations as $(s \doteq t)\sigma = (s\sigma \doteq t\sigma)$, to sets as $\Delta\sigma = \biguplus_{e\in\Delta}\{e\sigma\}$, and to pairs as $\langle r_1, r_2\rangle\sigma = \langle r_1\sigma, r_2\sigma\rangle$. The *domain of a substitution* σ, denoted $\mathtt{dom}(\sigma)$, is defined as usual, i.e. $\mathtt{dom}(\sigma) = \{z \in \mathcal{V} \mid z\sigma \neq z\}$. The *composition* of σ and θ, denoted $\theta \circ \sigma$, is defined as $\{\alpha \mapsto \alpha\sigma\theta \mid \alpha \in \mathtt{dom}(\sigma) \cup \mathtt{dom}(\theta)\}$. For substitutions σ, θ, $\sigma = \theta$ holds if $\forall z \in \mathcal{V} : z\sigma = z\theta$. Moreover, σ is *more general* than θ, denoted $\sigma \leq \theta$, if there exists η such that $\theta = \eta \circ \sigma$.

A *unifier* of two terms s, t is a substitution σ such that $s\sigma = t\sigma$. A unifier does not always exist. We capture that situation by simply saying that the unifier of s and t is $\bot$. We define the *most general unifier* of two *first-order* terms s and t, denoted $\mathtt{mgu}(s = t)$, as *any* unifier σ of s and t such that, for every unifier θ of s and t, $\sigma \leq \theta$ holds. If such substitution does not exist we say that $\mathtt{mgu}(s = t)$ is *not defined*, denoted $\mathtt{mgu}(s = t) = \bot$. In an abuse of notation, we assume that $t\sigma = \bot$ for every term t if $\sigma = \bot$ and extend the definitions for the application of a substitution on a term, equation, set of equations, and list of terms accordingly.

First-order unification can be performed in polynomial time, if the algorithm works on a shared term representation. We assume that terms are represented as DAGs, which allows sharing of common subterms. When using DAGs, performing unification and applying substitutions takes polynomial time. However, for example visiting all positions in term may take worst-case exponential time.

3 One Context Unification

The One Context Unification Problem (1CU) consists on finding a unifier for a set of equations over first-order terms that are extended with a single context variable F. It is known that the problem is in NP. It is also known that, if the set of equations contains an equation of the form $F(r_1) \doteq CF(r_2)$ where C is not the empty context, then unification can be performed

in polynomial time. Thus, without loss of generality, we can focus on 1CU instances of the form

$$\{\ F(r_1) \doteq s_1,\ \ \ldots,\ \ F(r_n) \doteq s_n\ \}$$

where s_i and r_i do not contain occurrences of F, for all $i \in \{1, \ldots, n\}$.

▶ **Definition 3.1.** A 2-restricted 1CU instance $\mathcal{I}$ (referred to by 1CU2r) consists of two equations of the form

$$F(r_1) \doteq s,\ F(r_2) \doteq t$$

where F is a context variable and s, t, r_1, r_2 are terms that may contain first-order variables but not the context variable.

The size of $\mathcal{I}$, denoted $||\mathcal{I}||$, is the size of the DAG representing all the terms (including subterms) in $\mathcal{I}$. In other words, $||\mathcal{I}|| = |\mathtt{subterms}(\mathcal{I})|$. From now on, when we refer to a polynomial time algorithm for 1CU2r, we implicitly assume this measure.

A solution, or unifier, σ of a 1CU2r instance can be characterized by $\mathtt{hp}(F\sigma)$ – the position of the hole in $F\sigma$.

▶ **Example 3.2.** The instance $\mathcal{I} = \{F(a) \doteq f(x_0, x_0),\ F(b) \doteq f(f(x_1, x_1), f(a, b))\}$ has a solution $\sigma = \{x_0 \mapsto f(a, a),\ x_1 \mapsto a,\ F \mapsto f(f(a, a), f(a, \bullet))\}$. Here, $\mathtt{hp}(F\sigma) = 2.2$. There is another solution with hole position 1.1.

In the first part of this paper we solve the decision version of the 1CU2r problem and later we show how to compute a representation for all unifiers.

4 The Decision Version of 1CU2r

We present a polynomial time algorithm that decides the 1CU2r unification problem. We describe our algorithm using *inference rules* that operate on *states*. Starting from an *initial state* that describes the input 1CU2r instance, the algorithm works by repeatedly applying the inference rules until a *final* state (where no rule is applicable) is reached. The final state will be a special fail state if the input problem has no unifiers.

4.1 Defining the state

Our inference rules operate on states (configurations), which are tuples of the form (R, Δ), where R is a pair of terms and Δ is a set of equations $s_1 \doteq t_1, \ldots, s_n \doteq t_n$, where each equation can be asymmetric (unmarked) or symmetric (marked with a superscript S).

▶ **Definition 4.1** (Initial state)**.** For a given instance $\mathcal{I} = \{F(r_1) \doteq s, F(r_2) \doteq t\}$ of the 1CU2r problem, the initial state of our algorithm is $\mathcal{S}_0 = (\langle r_1, r_2\rangle, \{s \doteq t\})$.

All terms in a state are first-order terms. The terms in R are the left-hand side terms and the equations in Δ are (different possible) right-hand side terms. While $u \doteq v$ and $v \doteq u$ are different equations, we do not distinguish between $u \doteq^S v$ and $v \doteq^S u$. If we write $x = s$ without a dot, then this is only a notation for "either $x \doteq s$ or $x \doteq^S s$".

The reason for keeping the right-hand side terms s, t as an equation $s \doteq t$ is that our inference rules will be trying to (almost) first-order unify s and t, but for one position (which will be the hole position of F in the solution).

The mapping from a state to 1CU2r instances is defined as follows.

▶ **Definition 4.2.** Let $\mathcal{S} = (\langle r_1, r_2\rangle, \Delta)$ be a state, and let $e = (s =^S t) \in \Delta$ be a *symmetric* equation. Let $\theta = \mathtt{mgu}(\Delta \setminus \{e\})$. We define the two 1CU instances spanned by e in $\mathcal{S}$, denoted $P(e, \mathcal{S}, 1)$ and $P(e, \mathcal{S}, 2)$ (or simply $P(e, i)$ if $\mathcal{S}$ is clear from the context), as

$$\begin{array}{rcll} P(e,\mathcal{S},1) & = & \{F(r_1\theta) \doteq s\theta,\ F(r_2\theta) \doteq t\theta\} & \text{if } \theta \neq \bot \\ P(e,\mathcal{S},2) & = & \{F(r_2\theta) \doteq s\theta,\ F(r_1\theta) \doteq t\theta\} & \text{if } \theta \neq \bot \\ P(e,\mathcal{S},i) & = & \bot & \text{if } \theta = \bot, \text{ for } i = 1,2 \end{array}$$

For an asymmetric equation $e \in \Delta$, $P(e, \mathcal{S}, 1)$ is defined as it is defined for symmetric equations, but $P(e, \mathcal{S}, 2) = \bot$ always. Given a state $\mathcal{S} = (L, \Delta)$ and a subset $\Gamma \subseteq \Delta$, by $\mathtt{instances}(\Gamma)$ we denote the set $\bigcup_{e\in\Gamma}(\{P(e, \mathcal{S}, 1), P(e, \mathcal{S}, 2)\})$.

Note that every equation e in the set Δ of a state spans zero, one or two 1CU2r instances, depending on whether $\theta = \bot$ and whether e is symmetric. Note that the initial state for a 1CU2r instance spans that 1CU2r instance. We say that a state has a solution if one of the instances that it spans is unifiable.

4.2 An Illustrative Example

We illustrate our procedure on a simple example before presenting it formally. Consider the instance $\mathcal{I}$ from Example 3.2. The corresponding initial state is

$$\mathcal{S}_0 \quad = \quad (\langle a, b\rangle, \{f(x_0, x_0) \doteq f(f(x_1, x_1), f(a, b))\})$$

The idea behind our procedure is that it searches for a solution σ by searching for $\mathtt{hp}(F\sigma)$ – call it p. For this example, the value $p = \lambda$ (i.e., $F \mapsto \bullet$) does not work, and hence, we need to find if $p = 1.p'$ or $p = 2.p'$ for some p'. So, we "decompose" (as in first-order unification) the equation in Δ to get a new state

$$\mathcal{S}_1 \quad = \quad (\langle a, b\rangle, \{x_0 \doteq f(x_1, x_1),\ x_0 \doteq f(a, b)\})$$

Let us construct the two instances $\mathcal{I}_1, \mathcal{I}_2$ corresponding to the state $\mathcal{S}_1$.

$$\begin{array}{rcl} \mathcal{I}_1 & = & P(x_0 \doteq f(x_1, x_1), 1) \ = \ \{F(a) \doteq f(a, b),\ F(b) \doteq f(x_1, x_1)\} \\ \mathcal{I}_2 & = & P(x_0 \doteq f(a, b), 1) \ = \ \{F(a) \doteq f(x_1, x_1),\ F(b) \doteq f(a, b)\} \end{array}$$

Note that $\mathcal{I}$ has a solution iff $\mathcal{I}_1$ or $\mathcal{I}_2$ has a solution.

A natural way to proceed would be to solve $\mathcal{I}_1$ and $\mathcal{I}_2$ recursively. However, that approach may perform an exponential number of steps since it is visiting all positions in a term that is represented as a DAG. Instead, our algorithm, roughly speaking, solves $\mathcal{I}_1$ and $\mathcal{I}_2$ simultaneously using a "Merge rule", which generates state $\mathcal{S}_2$ from the state $\mathcal{S}_1$.

$$\mathcal{S}_2 \quad = \quad (\langle a, b\rangle, \{f(x_1, x_1) \doteq^S f(a, b)\}).$$

Note that, again by Definition 4.2, $\mathcal{S}_2$ spans $\mathcal{I}_1$ and $\mathcal{I}_2$, since $P(f(x_1, x_1) \doteq^S f(a, b), \mathcal{S}_2, 1) = \mathcal{I}_2$ and $P(f(x_1, x_1) \doteq^S f(a, b), \mathcal{S}_2, 2) = \mathcal{I}_1$, and hence we have not lost any solutions. Note that the symmetric mark indicates that we need to try both orientations of the equation.

We can continue by applying "decompose" to $\mathcal{S}_2$ to get $\mathcal{S}_3$:

$$\mathcal{S}_3 \quad = \quad (\langle a, b\rangle, \{x_1 \doteq^S a, x_1 \doteq^S b\}).$$

And we can again use the "Merge rule" to get $\mathcal{S}_4$:

$$\mathcal{S}_4 \quad = \quad (\langle a, b\rangle, \{a \doteq^S b\}).$$

$$\texttt{Decompose:}\quad \frac{\langle R, \Delta = \Gamma \cup \Delta' \rangle}{\texttt{SolveFO}(\mathcal{I}_1\sigma) \mid \ldots \mid \texttt{SolveFO}(\mathcal{I}_{2|\Gamma|}\sigma) \mid \texttt{Decompose}(\langle R, \Delta\rangle, \Gamma, \mathcal{Y})}$$
where $\bigcup \mathcal{I}_i = \texttt{instances}(\Gamma)$, $\sigma = \{F \to \bullet\}$, $|\texttt{topsymbols}(\Gamma)| = 1$, and $\Gamma \subset \Delta$ is a root class.

$$\texttt{Easy:}\quad \frac{\langle R, \Delta \cup \{e\}\rangle}{\texttt{SolvEz}(P(e,1)) \mid \texttt{SolvEz}(P(e,2)) \mid \langle R\,\texttt{mgu}(e), \Delta\,\texttt{mgu}(e)\rangle} \quad \text{if } P(e,1) \text{ is easy}$$

$$\texttt{InvEq:}\quad \frac{\langle R, \Delta\rangle}{\langle R\sigma, \Delta\sigma\rangle} \quad \text{if } \Delta \models (s = t) \text{ and } \sigma = \texttt{mgu}(s \doteq t)$$

$$\texttt{DiscardEq:}\quad \frac{\langle R, \Delta \cup \{e\}\rangle}{\langle R\,\texttt{mgu}(e), \Delta\,\texttt{mgu}(e)\rangle} \quad \text{if } \texttt{mgu}(\Delta) = \bot$$

$$\texttt{Merge:}\quad \frac{\langle R, \Delta \cup \{x = s, x = t\}\rangle}{\langle R, \Delta \cup \{s \doteq^S t\}\rangle} \quad \text{if } x \notin \texttt{vars}(\Delta) \text{ and } x \notin \texttt{vars}(R)$$

$$\texttt{Merge:}\quad \frac{\langle R, \Delta \cup \{x = s, t = x\}\rangle}{\langle R, \Delta \cup \{t \doteq^? s\}\rangle} \quad \text{if } x \notin \texttt{vars}(\Delta) \text{ and } x \notin \texttt{vars}(R)$$
where ? is S iff $x = s$ or $t = x$ is symmetric.

$$\texttt{Fail:}\quad \frac{\langle \bot, \bot\rangle}{fail}$$

Figure 1 Inference rules for deciding 2-restricted one context unification.

We observe that the instance $P(a \doteq^S b, \mathcal{S}_4, 1)$ has a unifier that maps the context variable to $\bullet$, and we can terminate declaring that the original problem $\mathcal{I}$ has a unifier.

Instead of $\mathcal{I}$, if we start with the instance $\mathcal{I}' = \{F(a) \doteq f(x_0, f(a,b)), F(b) \doteq f(f(x_1, x_1), x_0)\}$, then after decomposing the initial state S_0' we would get the state $\mathcal{S}_1'$ containing the equations $x_0 \doteq f(x_1, x_1),\ f(a,b) \doteq x_0$ in its Δ component. Now, the two instances $\mathcal{I}_1', \mathcal{I}_2'$ corresponding to this new state would be

$$\mathcal{I}_1' = P(x_0 \doteq f(x_1,x_1), \mathcal{S}_1', 1) = \{F(a) \doteq f(a,b),\ F(b) \doteq f(x_1,x_1)\}$$
$$\mathcal{I}_2' = P(f(a,b) \doteq x_0, \mathcal{S}_1', 1) = \{F(a) \doteq f(a,b),\ F(b) \doteq f(x_1,x_1)\}$$

These two instances are identical! In this case, we again apply "merge" on $\mathcal{S}_1'$, but we now get a new state $\mathcal{S}_2'$ with an *asymmetric* equation:

$$\mathcal{S}_2' = (\langle a, b\rangle, \{f(a,b) \doteq f(x_1,x_1)\}).$$

We "decompose" $\mathcal{S}_2'$ to get $\mathcal{S}_3'$ with equations $a \doteq x_1, b \doteq x_1$ (both asymmetric this time), but when we then apply "merge" on $\mathcal{S}_3'$, since x_1 is on the same side of the two equations, we generate a *symmetric* equation in $\mathcal{S}_4'$:

$$\mathcal{S}_4' = (\langle a, b\rangle, \{a \doteq^S b\})$$

Since one of the instances spanned by $\mathcal{S}_4'$ has a unifier, we know $\mathcal{I}'$ too has a unifier.

4.3 Inference Rules for Deciding 1CU2r

The inference rules for solving the decision version of 1CU2r are presented in Figure 1. The rule `Decompose` and the rule `Easy` generate more than one state (separated by |). However, all but one of the child states is immediately evaluated to $\top$ or $\bot$, and there is only one main child (shown last) along which the search proceeds (if all others evaluate to $\bot$).

Since we are working on a DAG representation for Δ, it is useful to keep in mind the graph representing the set Δ.

▶ **Definition 4.3.** Let $\mathcal{S} = (R, \Delta)$ be a state. Let V be the nodes of the DAG representing the terms (and subterms) in Δ. By $\equiv$ we denote the set of undirected edges $\{(u, v) \mid \exists u = v \in \Delta\}$. By $\rightarrow$ we denote the set of subterm edges $\{(u, v) \mid u = f(\ldots, v, \ldots) \text{ for some } f\}$. By $G(\Delta)$ we denote the graph with nodes V and the two kinds of edges $\equiv$ and $\rightarrow$.

We describe the individual inference rules below.

4.3.1 The `Decompose` rule

As mentioned above, the `Decompose` rule generates several states. However, all of them but one correspond to first-order unification instances that are solved immediately in polynomial time using a procedure `SolveFO`.

As in the Paterson and Wegman linear unification [19] algorithm, the `Decompose` rule first finds a *root class* $\Gamma \subseteq \Delta$ in the graph $G(\Delta)$ (Definition 4.3). A root class is a maximum cardinality subset Γ of Δ such that if t is a topterm in Γ, then (a) t is not a proper subterm of any term in Δ, and (b) t is not a topterm in $\Delta - \Gamma$. If such a nonempty $\Gamma = \{e_1, \ldots, e_{|\Gamma|}\}$ exists, and if all non-variable terms in Γ have top symbol f, then the `Decompose` rule can be applied. First, we instantiate the variables in Γ. Let $\theta = \{x \rightarrow f(y^x_1, \ldots, y^x_{ar(f)}) \mid x \in \texttt{topvars}(\Gamma), y^x_i \text{ are fresh variables from } \mathcal{Y}\}$. Note that all terms in $\Gamma\theta$ will have f as their top symbol.

▶ **Definition 4.4** (Decompose). Given a state $\mathcal{S} = (R, \Delta)$, we define $\texttt{Decompose}(\mathcal{S}, \Gamma, \mathcal{Y})$ as the state $\mathcal{S}' = (R\theta, \Delta')$ where $\Delta' = (\Delta - \Gamma) \cup \Gamma'$ and Γ' is obtained from $\Gamma\theta$ as follows:

1. If there is an asymmetric equation $e = (f(u_1, \ldots, u_k) \doteq f(v_1, \ldots, v_k))$ in $\Gamma\theta$, then replace e by several equations $u_1 \doteq v_1, \ldots, u_k \doteq v_k$.
2. If there is a symmetric equation $e = (f(u_1, \ldots, u_k) \doteq^S f(v_1, \ldots, v_k))$ in $\Gamma\theta$, then replace e by several equations $u_1 \doteq^S v_1, \ldots, u_k \doteq^S v_k$.

Note the differences with the decompose rule in classical first-order unification: in first-order unification, decomposition is not performed when there are variables in an equivalence class. Moreover, we restrict our application of the rule to a root class as in Paterson and Wegman [19]. Conditions (a) and (b) above ensure the invariant that variables from $\mathcal{Y}$ occur only as topterms in the equations Δ. This fact will be crucial to prove termination in polynomial time.

4.3.2 The Shrink rules

The rules `Easy`, `InvEq`, and `DiscardEq` remove at least one equation from the set Δ and help in guaranteeing that the size of Δ stays polynomial w.r.t. the input size.

4.3.2.1 The `Easy` rule

We first enumerate some easily solvable instances of 1CU2r.

▶ **Definition 4.5** (Easy). A 1CU2r instance $\{F(r_1) \doteq s, F(r_2) \doteq t\}$ is *easy* if either one of the following condition holds: (a) $t = C[s]$ (or $s = C[t]$) for some non-empty context C, or (b) $\texttt{topsymbol}(s) \neq \texttt{topsymbol}(t)$, or (c) $s = t$, or (d) $s \in \mathcal{X}$ or $t \in \mathcal{X}$, or (e) s (or t) occurs (as a subterm) in r_1 or r_2.

▶ **Lemma 4.6** (Easy). *There is a polynomial time procedure* `SolvEz` *that returns* $\top$ *(denoting* `True`*) iff a given easy instance* $\mathcal{I}$ *is unifiable.*

We will discuss easy instances in more detail in Section 6. For a symmetric equation e, note that $P(e,1)$ is easy iff $P(e,2)$ is easy. The rule `Easy` checks if $P(e,1)$ is easy for some $e \in \Delta$ and solves it immediately. If both $P(e,i)$ are found to be non-unifiable, then the search proceeds with the new state obtained by removing e from Δ. Note that e is removed by applying $\texttt{mgu}(e)$ to the rest of Δ and R, and recall that if $\texttt{mgu}(e) = \bot$ then $\Delta\,\texttt{mgu}(e) = \bot$.

4.3.2.2 The `InvEq` rule

If there are terms s and t that are made equal by every unifier of a 1CU2r instance, then we can as well unify s and t and apply the unifier to the 1CU2r instance. The rule `InvEq` performs this step. Let $\mathcal{S} = (R, \Delta)$ be a state. We write $\Delta \models (s = t)$ if s and t are distinct terms that lie in an $\equiv$-cycle of $G(\Delta)$ (cycle consisting solely of $\equiv$ edges). It should be evident that if $\Delta \models (s = t)$, then for all $e \in \Delta$, $s\sigma = t\sigma$ holds for mgu σ of $\Delta \setminus \{e\}$; that is, s and t are made equal by every unifier of any instance spanned by the state $\mathcal{S}$.

4.3.2.3 The `DiscardEq` rule

The rule `DiscardEq` removes an equation e from Δ if the problems $P(e, \mathcal{S}, i)$ spanned by e have no solutions. This happens, for example, when $\texttt{mgu}(\Delta \setminus \{e\}) = \bot$, by Definition 4.2.

4.4 The `Merge` rule

Recall that the `Decompose` rule instantiates variables in the root class to enable the decomposition. In one special case, we prefer to apply the `Merge` rule rather than the `Decompose` rule. Specifically, the `Merge` rule is applicable to state (R, Δ) if the root class $\Gamma \subset \Delta$ contains a variable x such that (a) x occurs exactly twice in $\texttt{topterms}(\Gamma)$ and (b) x does not occur in the terms of R.

In such a case, $\Delta = \Delta' \cup \{e_1, e_2\}$, where e_1, e_2 might be of the form:

1. $e_1 = (x = s_1)$ and $e_2 = (x = s_2)$, or $e_1 = (s_1 = x)$ and $e_2 = (s_2 = x)$: In these cases, the two equations e_1, e_2 are replaced by a single symmetric equation $s_1 \doteq^S s_2$.
2. $e_1 = (s_1 = x)$ and $e_2 = (x = s_2)$: In this case, e_1, e_2 are replaced by $s_1 \doteq s_2$ if both e_1 and e_2 are asymmetric, and by $s_1 \doteq^S s_2$ otherwise.

The `Merge` rule captures one of the key insights in our procedure: we can simultaneously search for solutions for $\{F(r_1) \doteq s, F(r_2) \doteq t\}$ and $\{F(r_1) \doteq t, F(r_2) \doteq s\}$, and this commutativity is the only source of blowup for 1CU2r (see also the example in 4.2). While most of the rules presented so far have some similarity to the rules for more general 1CU problems that we presented in [9], there is no analogue of the merge rule in [9].

4.5 Correctness

We fix a strategy for applying the rules. Specifically, we apply the shrink rules and the `Merge` rules exhaustively (denoted by "!"), and apply `Decompose` only when none of the other rules are applicable.

$$((\text{Shrink} \mid \text{Merge})^!(\text{Decompose}))^!$$

To provide intuition for the working of our procedure, consider the graph $G(\Delta)$ with $\equiv$ and $\rightarrow$ edges (Definition 4.3). Either there is a cycle in this graph or there is none. If there is an $\equiv$-cycle, then we can apply `InvEq` rule to eliminate it. If there is a cycle that goes through an $\rightarrow$ (subterm) edge, (which corresponds to an occurs-check violation in first-order unification), then the inference rule `Easy` will be applicable (since for some edge

$s \doteq t$ in the cycle, we will have $s = C[t]$ or $t = C[s]$ for some nonempty context C, which is Case (a) in Definition 4.5). We exhaustively apply shrink rules, and since they reduce $|\Delta|$ or $|\texttt{topterms}(\Delta)|$, we will eventually stop. When none of the shrink rules are applicable, then it means there are no cycles in $G(\Delta)$. This implies that there will be a root class Γ. If there are two different top symbols, say f and g, among the terms in Γ, then we would apply the `Easy` rule first (some instance spanned by the state will satisfy Case (b) of Definition 4.5). If Γ contains only equations between variables, then these variables can not occur anywhere else in Δ (because Γ is the root class), and hence, again `Easy` would be applicable (some instance spanned by the state will satisfy Case (d) of Definition 4.5). Hence, we conclude that Γ contains at least one non-variable term and all non-variable terms will have the same function symbol at the top. As a result, the `Decompose` rule will be applicable. If `Merge` is applicable, we first apply that rule and finally, when no more `Merge` applications are possible, we apply `Decompose`, and repeat.

We say that a state has a solution if one of its spanned instances (Definition 4.2) has a solution. It is easily verified that each inference rule preserves the existence of a solution, and it does not introduce any new solutions. Furthermore, since the above argument guarantees progress (stated in Lemma 4.7), correctness follows.

▶ **Lemma 4.7.** *Let $\mathcal{S} = (R, \Delta)$ be a valid state of our algorithm. If $\Delta \neq \emptyset$, then a rule can be applied to $\mathcal{S}$.*

4.6 Termination

Without loss of generality, we assume that the DAG D representing all the terms (and subterms) in any state are minimal in size, and hence the correspondence between terms and nodes is bijective. Hence, we can then refer to nodes of D and subterms of the problem as if they were the same thing. In the rest of this section, when we obtain a bound w.r.t. $\texttt{subterms}(\texttt{topterms}(\Delta))$, for some set of equations Δ, the bound directly translates to the size of the DAG D representing the terms in Δ. A crucial observation is that the *number* of terms represented in a DAG is preserved by the application of substitutions resulting from the unification of terms of the DAG. This is because application of such substitutions can be achieved by manipulating only the *edges* of the DAG, leaving its nodes untouched.

Our algorithm reduces deciding solvability of a 1CU2r instance $\mathcal{I}$ to solving instances $\mathcal{I}_1, \ldots, \mathcal{I}_n$ where each $\mathcal{I}_i$ is either an *easy* instance (generated by rule `Easy`) or a first-order unification instance (generated by rule `Decompose`). Hence, to prove that our algorithm terminates in polynomial time we have to argue that (A) $\mathcal{I}_i$ has polynomial size, for all i, and that (B) n is polynomial.

One of the main difficulties in the proof is due to the fact our inference system introduces fresh variables from $\mathcal{Y}$. We first show that the rule applications do not increase the total number of subterms in the equations of Δ that are not variables from $\mathcal{Y}$. Moreover, such a measure decreases with every application of `Decompose`.

▶ **Lemma 4.8.** *Let $(R, \Delta_0) \to^* (R_k, \Delta_k)$ be a derivation starting from a valid initial state. Then, $|\texttt{subterms}(\texttt{topterms}(\Delta_k) \setminus \mathcal{Y})| \leq |\texttt{subterms}(\texttt{topterms}(\Delta_0)|$. Moreover, if $(R, \Delta_0) \to^* (R_k, \Delta_k) \to_{\texttt{Decompose}} (R_{k+1}, \Delta_{k+1})$ then $|\texttt{subterms}(\texttt{topterms}(\Delta_{k+1})) \setminus \mathcal{V}| < |\texttt{subterms}(\texttt{topterms}(\Delta_k)) \setminus \mathcal{V}|$.*

Proof. The lemma follows by inspecting the rules, the observation above that the total number of subterms from $\mathcal{T}(\mathcal{F}, \mathcal{X})$ in every Δ_i is preserved by the application of substitutions resulting from the unification of terms from $\mathcal{T}(\mathcal{F}, \mathcal{X})$, and the assumption, w.l.o.g., that

variables from $\mathcal{Y}$ are always instantiated in terms of variables from $\mathcal{X}$. The second part of the lemma follows from the maximality of Γ in the `Decompose` rule. ◀

Let us call a state minimized if the shrink and merge rules are not applicable. We now prove a bound on the cardinality of Δ in minimized states that is independent of the number of (old and new) variables in the equations.

▶ **Lemma 4.9.** *Let (R, Δ) be a minimized state. Then, $|\Delta| \leq 2|\mathit{topterms}(\Delta) \setminus \mathcal{V}|$.*

Proof. Consider the subgraph G' of $G(\Delta)$ (Definition 4.3) defined by only the $\equiv$ edges. Nodes of G' in $\mathcal{V}$ are called v-nodes, while the rest are called f-nodes. Note that, since Δ is minimized, the following holds: (i) G' is acyclic (by non-applicability of `Easy`), (ii) Every v-node in G' has degree at least 3 and each subtree hanging off v has at least one f-node (by non-applicability of `Easy` and `Merge`). We prove that for any forest that satisfies (i) and (ii), $n \leq 2m - 2$, where n is the number of nodes the forest and m is the number of f-nodes in it. For simplicity, and w.l.o.g., assume that G' only has one connected component and hence it is a tree. We use induction on m. (Base case) If $m = 2$, then $n = 2$ and the claim holds. (Induction step) Let $m > 2$. If there are no v-nodes in G', then $n = m > 2$ and the claim holds. So, let v be a v-node in G' connected to the disjoint subtrees $\{t_1, \ldots, t_l\}$. Note that $l \geq 3$. Let n_i and m_i be the number of nodes and f-nodes in tree t_i, respectively, for all i. Note that $m_i > 0$. Consider tree t_i with node v attached to it. Call it t'_i. If we treat v as an f-node in t'_i, then it satisfies (i) and (ii) and we can apply induction hypothesis to get $n_i + 1 \leq 2(m_i + 1) - 2$; that is, $n_i \leq 2m_i - 1$. Hence we have $n = \sum_{i=1}^{l}(n_i) + 1 \leq \sum_{i=1}^{l}(2m_i - 1) + 1 = 2m - l + 1 \leq 2m - 2$, where the last step holds because $l \geq 3$. Hence, since the total number of nodes in G' is less than $2|\mathtt{topterms}(\Delta) \setminus \mathcal{V}|$, so is its number of edges and thus $|\Delta|$. ◀

We can now extend the previous claim to any state. The following lemma relies on the rule application strategy.

▶ **Lemma 4.10.** *Let $(R_0, \Delta_0) \to^* (R_k, \Delta_k)$ be a derivation starting from a valid initial state. Then, $|\Delta_k| \leq 2|\mathit{topterms}(\Delta_k) \setminus \mathcal{V}|\mathtt{maxarity}$.*

Proof. We use induction on k. The lemma trivially holds for Δ_0, since it only contains one equation and we can assume w.l.o.g. that such equation contains a non-variable term. For the inductive step, note that the property of the lemma is trivially preserved by all the rules but `Decompose`, since such rules do not increase the size of the Δ component of the state. For the `Decompose` rule, recall that this rule is only applicable to a minimized state Δ. By Lemma 4.9, Δ satisfies $|\Delta| \leq 2|\mathtt{topterms}(\Delta) \setminus \mathcal{V}|$. Hence, the lemma follows from the fact that `Decompose` increases the number of equations in Δ by a factor of, at most, `maxarity`. ◀

We can finally prove claims (A) and (B) required for polynomial time termination.

▶ **Lemma 4.11.** *Let $\mathcal{I}$ be a 1CU2r instance and let $(R_0, \Delta_0) \to^* \mathcal{S}_k = (R_k, \Delta_k)$ be its corresponding derivation. Then, $k \leq 4||\mathcal{I}||^2\mathtt{maxarity}$ and, for every equation $e \in \Delta_k$, $P(e, \mathcal{S}_k, 1)$ and $P(e, \mathcal{S}_k, 2)$ have polynomial size with respect to $||\mathcal{I}||$.*

Proof. By Lemma 4.8, $\forall i \in \{1, \ldots, k\} : |\mathtt{subterms}(\mathtt{topterms}(\Delta_i) \setminus \mathcal{Y})| \leq |\mathtt{subterms}(\mathtt{topterms}(\Delta_0))|$, i.e., the number of non-variable subterms in the Δ_is in the derivation does not increase. By Lemma 4.10, we have that every Δ_i in the derivation satisfies $|\Delta_i| \leq 2|\mathtt{topterms}(\Delta_i) \setminus \mathcal{V}|\mathtt{maxarity}$. Since $|\mathtt{topterms}(\Delta_i) \setminus \mathcal{V}| \leq |\mathtt{subterms}(\mathtt{topterms}(\Delta_i) \setminus \mathcal{Y})|$, we

have the bound $|\Delta_i| \leq 2|\mathtt{subterms}(\mathtt{topterms}(\Delta_0))|\mathtt{maxarity} \leq 2||\mathcal{I}||\mathtt{maxarity}$. For the first statement of the lemma, note that subsequences of rule applications without the `Decompose` rule have length at most $2|\Delta_i| \leq 4||\mathcal{I}||\mathtt{maxarity}$, since the rules in such subsequences either reduce the cardinality of Δ or $\mathtt{topterms}(\Delta)$. On the other hand, by Lemma 4.8, each application of `Decompose` decreases the measure $|\mathtt{subterms}(\mathtt{topterms}(\Delta_i) \setminus \mathcal{Y})| \leq |\mathtt{subterms}(\mathtt{topterms}(\Delta_0))| \leq ||\mathcal{I}||$. Hence, we have $k \leq 4||\mathcal{I}||^2\mathtt{maxarity}$. The second statement of the lemma follows from the fact that k is polynomial and that DAGs are used for term representation. ◀

5 Representing All Unifiers

We now extend the procedure for deciding 1CU2r to also output a representation for all unifiers. The procedure works in the same way, except that now we do not terminate when we find an instance that is unifiable (in Rules `Decompose` and `Easy`), but rather we store the representation for all solutions returned by the subroutines and continue.

Given a 1CU2r instance $\mathcal{I}$, rather than returning all possible unifiers σ of $\mathcal{I}$, our algorithm will only return the positions $\mathtt{hp}(F\sigma)$. This is not a loss of generality, since σ can be recovered from $\mathtt{hp}(F\sigma)$ and $\mathcal{I}$ in polynomial time.

▶ **Definition 5.1** (Sets of solutions)**.** Let $\mathcal{I}$ be a 1CU2r instance. The set $\mathtt{HP}(\mathcal{I})$ of solutions of $\mathcal{I}$ is the set of positions $\{\mathtt{hp}(F\sigma) \mid \sigma \text{ is a solution of } \mathcal{I}\}$.

Using $\mathtt{HP}(\mathcal{I})$ as a substitute for all unifiers simplifies presentation and notation considerably. However, the cardinality of $\mathtt{HP}(\mathcal{I})$ may grow exponentially, or even be unbounded.

▶ **Example 5.2.** Let s be $f(x_0, x_0)$ and let $t_0 = f(a, b)$, $t_1 = f(f(x_1, x_1), f(a, b))$, and in general, let $t_n = f(f(x_n, x_n), t_{n-1})$ for $n \geq 1$. If $\mathcal{I} = \{F(z_1) \doteq s, F(z_2) \doteq t_n\}$ where z_1, z_2 are fresh, then the set $\mathtt{HP}(\mathcal{I})$ contains all positions with length at most $n + 1$, and hence, its cardinality is exponential in n. If $\mathcal{I} = \{F(a) \doteq s, F(b) \doteq t_n\}$, then $\mathtt{HP}(\mathcal{I})$ contains all positions of length n that contain an even number of 1s. These instances have an exponential worst-case running time for the algorithms in [8]. Our algorithm will represent the sets $\mathtt{HP}(\mathcal{I})$ in polynomial space by means of *abstract positions*.

Our algorithm will construct a succinct representation, using a grammar-formalism and also exponentiation, for $\mathtt{HP}(\mathcal{I})$, satisfying that: (1) a solution of $\mathcal{I}$ can be obtained in polynomial time, (2) the number of solutions m (or whether it is infinite) of $\mathcal{I}$ can be found (or decided) in polynomial time, and (3) all solutions of $\mathcal{I}$ can be obtained in polynomial time w.r.t. m.

To keep track of all solutions, the states over which our inference system works will be tuples of the form (R, Δ, S), where R is a pair of terms, Δ is a set of *labeled*, possibly marked (with S), equations $s_1 \doteq_{l_1} t_1, \ldots, s_n \doteq_{l_n} t_n$, where no label l_i occurs more than once, and S is a set of (*representations* for) set of positions. For an instance $\mathcal{I} = \{F(r_1) \doteq s, F(r_2) \doteq t\}$, the initial state will now be $(\langle r_1, r_2\rangle, \{s \doteq_\lambda t\}, \emptyset)$.

5.1 The Labels and the Representation of Solutions

The set L of labels in the equations Δ are terms generated using the following BNF grammar:

$$L \quad ::== \quad i \mid L.L \mid L + L \mid L \oplus L$$

where i denotes a number in $\{1, \ldots, \mathtt{maxarity}\}$. We call elements of L *abstract positions*. Note that L contains all concrete positions (that is, $\mathtt{pos}(\mathcal{F}) \subset L$). Each abstract position

denotes a set of concrete positions, but we delay the formal definition of the mapping until later. Intuitively, $\oplus$ and $+$ are used to distinguish applications of the `Merge` rule that produce a symmetric equation from those that do not. This information is necessary to correctly recover all solutions in our representation.

The elements of the third component of the state, which represent a set of solutions, are strings in the set

$$L_S = \{\langle l, i\rangle \mid l \in L, i \in \{1,2\}\}^*.\texttt{BaseCases} \cup \{\lambda\}$$

where the set `BaseCases` will be defined later (in Section 5.3 and Section 6). Since expanding the labels may lead to exponentially large labels, the algorithm in fact uses only the nonterminals of a dynamically extended Straight-Line Program (SLP), similarly as done in [13] (see [18] for a survey on algorithms on SLP-compressed words), which keeps the size polynomial by sharing common prefixes. For simplicity, we obviate this representation in the definitions of the rules.

We assume that, for the easy instances (Definition 4.5), we have a procedure that returns some representation of the set of all solutions, which we have denoted by `BaseCases` above.

5.2 Modified Inference Rules

For each inference rule in Figure 1, we have to show how we update the labels in Δ and the third component of the state.

5.2.1 Modification for the Third Component

The only rules that change the third component are `Decompose` and `Easy`. We just need to compute the solutions for the easily solvable instances generated by these two rules, and add these solutions to the third component of the main branch.

▶ **Rule 5.3** (`Decompose`). *When applying* `Decompose` *to a state* $\mathcal{S} = (R, \Delta = \Gamma \cup \Delta', S)$*, the function* $\texttt{Decompose}(\langle R, \Delta\rangle, \Gamma, \mathcal{Y})$ *returns the state whose third component* S' *is obtained as follows:*

$S' = S$;
for $(s =_l t) \in \Gamma$ ***do***
  Compute $\sigma_i = \texttt{mgu}(P(s =_l t, \mathcal{S}, i)\{F \to \bullet\})$*, for* $i \in \{1,2\}$;
  $S' = S' \cup \{\langle l, i\rangle \mid i \in \{1,2\}, \sigma_i \neq \bot\})$;
end for

Note that σ_i *corresponds to the* `mgu` *of the* i*th first-order unification problem resulting from the application of the* `Decompose` *rule in Figure 1.*

▶ **Rule 5.4** (`Easy`). *When applying* `Easy` *to a state* $\mathcal{S} = (R, \Delta \cup \{s \doteq_l t\}, S)$*, the third component* S' *of the last state generated by* `Easy` *is computed as follows:*

$$S' = S \cup \{\langle l, i\rangle.\texttt{SolvEz}(P(s \doteq_l t, \mathcal{S}, i)) \mid i \in \{1,2\}\}$$

where `SolvEz` *is now assumed to return a representation in* `BaseCases`*.*

5.2.2 Modifications for the Second Component

The only inference rules that add new equations in Δ are `Decompose` and `Merge`, and hence we need to specify how labels are assigned to these newly added equations. Labels on existing equations in Δ do not change.

The Decompose rule uses Definition 4.4 to generate the new state. In Definition 4.4, if the equation $e = (f(u_1, \ldots, u_k) \doteq_l f(v_1, \ldots, v_k))$ has label l, then the i-th generated equation, namely $u_i \doteq_{l.i} v_i$ is assigned label $l.i$. Labels on the symmetric variant in Definition 4.4 are assigned in the same way. The Merge rule introduces $+$ and $\oplus$ operators in the labels.

▶ **Rule 5.5** (Merge). *When applying* Merge *to a state* $\mathcal{S} = (R, \Delta \cup \{e_1, e_2\}, S)$*, if* e_1 *has label* l_1 *and* e_2 *has label* l_2*, then the generated equation has label* $l_1 \oplus l_2$ *if it is symmetric, and it has label* $l_1 + l_2$ *if it is asymmetric.*

The following example, derived from the family in Example 5.2 illustrates the goal of the Merge rule. Without it the algorithm would still be sound, but it would have worst-case exponential running time.

▶ **Example 5.6.** Consider the 1CU2r instance $\mathcal{I} = \{F(z_1) \doteq f(x_0, x_0), F(z_2) \doteq f(f(x_1, x_1), f(f(x_2, x_2), f(a, b)))\}$. The corresponding initial state of our algorithm is $\mathcal{S}_0 = (R = \langle z_1, z_2\rangle, \Delta = \{f(x_0, x_0) \doteq_\lambda f(f(x_1, x_1), f(f(x_2, x_2), f(a, b)))\}, S_0 = \emptyset)$, and we have the following derivation:
$\mathcal{S}_0 \to_{\texttt{Decompose}}$
$\mathcal{S}_1 = (R, \{x_0 \doteq_1 f(x_1, x_1), x_0 \doteq_2 f(f(x_2, x_2), f(a, b))\}, S_1 = \{\langle \lambda, 1\rangle\}) \to_{\texttt{Merge}}$
$\mathcal{S}_2 = (R, \{f(x_1, x_1) \doteq^S_{1\oplus 2} f(f(x_2, x_2), f(a, b))\}, S_1) \to_{\texttt{Decompose}}$
$\mathcal{S}_3 = (R, \{x_1 \doteq^S_{(1\oplus 2).1} f(x_2, x_2), x_1 \doteq^S_{(1\oplus 2).2} f(a, b)\}, S_2 = S_1 \cup \{\langle 1 \oplus 2, 1\rangle, \langle 1 \oplus 2, 2\rangle\}) \to_{\texttt{Merge}}$
$\mathcal{S}_4 = (R, \{f(x_2, x_2) \doteq^S_{((1\oplus 2).1)\oplus((1\oplus 2).2)} f(a, b)\}, S_2) \to_{\texttt{Decompose}}$
$\mathcal{S}_5 = (R, \{x_2 \doteq^S_{((1\oplus 2).1)\oplus((1\oplus 2).2).1} a, x_2 \doteq^S_{((1\oplus 2).1)\oplus((1\oplus 2).2).2} b\}, S_3 = S_2 \cup \{\langle((1 \oplus 2).1) \oplus ((1 \oplus 2).2), 1\rangle, \langle((1 \oplus 2).1) \oplus ((1 \oplus 2).2), 2\rangle\}) \to_{\texttt{Merge}}$
$\mathcal{S}_6 = (R, \{a \doteq^S_{(((1\oplus 2).1)\oplus((1\oplus 2).2).1)\oplus(((1\oplus 2).1)\oplus((1\oplus 2).2).1)} b\}, S_3) \to_{\texttt{Decompose}}$
$\mathcal{S}_7 = (R, \emptyset, S_4 = S_3 \cup \{\langle(((1 \oplus 2).1) \oplus ((1 \oplus 2).2).1 \oplus ((1 \oplus 2).1) \oplus ((1 \oplus 2).2).1, 1\rangle, \langle((1 \oplus 2).1) \oplus ((1 \oplus 2).2).1 \oplus ((1 \oplus 2).1) \oplus ((1 \oplus 2).2).1, 2\rangle\})$

This example also illustrates the need for a succinct representation for the elements in the set of solutions of the state. As mentioned in Section 5.1, we use a common grammar-like representation G that shares common prefixes for that purpose. Instead of defining such representation formally, we just extend the previous example by showing how S_4 looks like with the suitable representation: $S_4 = \{\langle N_0, 1\rangle, \langle N_1, 1\rangle, \langle N_1, 2\rangle, \langle N_4, 1\rangle, \langle N_4, 2\rangle, \langle N_7, 1\rangle, \langle N_7, 2\rangle\}$, where G is the set of rules $\{N_0 \to \lambda, N_1 \to (1 \oplus 2), N_2 \to N_1.1, N_3 \to N_1.2, N_4 \to (N_2 \oplus N_3), N_5 \to N_4.1, N_6 \to N_4.2, N_7 \to (N_5 \oplus N_6)\}$. This representation also allows us to update the set of solutions after a rule application in polynomial time.

5.3 Correctness

The strategy for applying the inference rules is unchanged. Note that the elements of the third component of the state are strings in the set L_S (defined in Section 5.1). We define $\texttt{BaseCases} = \{\texttt{exp}(p, e), \texttt{exp}(p, k_1 N + k_2), \texttt{hps}(u, v), p.free\}$, for a concrete position $p \in \texttt{pos}(\mathcal{F})$, natural numbers $e, k_1, k_2 \leq ||\mathcal{I}||$, and an integer variable N. Note that the elements of BaseCases are simply symbolic expressions that are succinct representations for sets of positions (as defined in Figure 2). The *concretization function* $\alpha : L_s \to \mathcal{P}(\texttt{pos}(\mathcal{F}))$, presented in Figure 2, maps elements in the third component of the state to a set of positions.

Before we can state the correctness claim that solutions are preserved by inference rule applications, we need to define the set of solutions of a state.

▶ **Definition 5.7** (Solution of a state). Let $\mathcal{S} = (R, \Delta, S)$ be a state. The set of solutions of $\mathcal{S}$, denoted $\texttt{HP}(\mathcal{S})$, is defined as $\texttt{HP}(\mathcal{S}) = \bigcup_{(s=_l t)\in\Delta} \bigcup_{i\in\{1,2\}} (\alpha(\langle l, i\rangle).\texttt{HP}(P(s =_l t, \mathcal{S}, i)))$

$$
\begin{aligned}
&\alpha(\langle i,1\rangle) = \{i\} \\
&\alpha(\langle l_1.l_2,1\rangle) = \big(\alpha(\langle l_1,1\rangle).\alpha(\langle l_2,1\rangle)\big) \cup \\
&\qquad\qquad \big(\alpha(\langle l_1,2\rangle).\alpha(\langle l_2,2\rangle)\big) \\
&\alpha(\langle l_1.l_2,2\rangle) = \big(\alpha(\langle l_1,1\rangle).\alpha(\langle l_2,2\rangle)\big) \cup \\
&\qquad\qquad \big(\alpha(\langle l_1,2\rangle).\alpha(\langle l_2,1\rangle)\big) \\
&\alpha(\mathtt{exp}(p,e)) = \{p^e\} \\
&\alpha(p.free) = \{q \mid q \geq p\}
\end{aligned}
\qquad
\begin{aligned}
&\alpha(\langle l_1 + l_2, j\rangle) = \alpha(\langle l_1,j\rangle) \cup \alpha(\langle l_2,j\rangle) \\
&\alpha(\langle l_1 \oplus l_2,1\rangle) = \alpha(\langle l_1,1\rangle) \cup \alpha(\langle l_2,2\rangle)\big) \\
&\alpha(\langle l_1 \oplus l_2,2\rangle) = \alpha(\langle l_1,2\rangle) \cup \alpha(\langle l_2,1\rangle)\big) \\
&\alpha(c_1.c_2) = \alpha(c_1).\alpha(c_2) \\
&\alpha(\mathtt{exp}(p,k_1N+k_2)) = \{p^{k_1k+k_2} \mid k \geq ||\mathcal{I}||^2\} \\
&\alpha(\mathtt{hps}(u,v)) = \{q \mid v|_q = u\}
\end{aligned}
$$

Figure 2 Definition of the concretization function α.

It it easy to see that, for a 1CU2r instance $\mathcal{I}$ and the corresponding initial state $\mathcal{S}$ (Definition 4.1), $\mathtt{HP}(\mathcal{I}) = \mathtt{HP}(\mathcal{S})$, by Definition 4.2. Hence, to show correctness it suffices to prove that for every rule application, $\mathcal{S} \to_r \mathcal{S}'$ of a rule r on a valid state $\mathcal{S}$, $\mathtt{HP}(\mathcal{S}) \cup \alpha(S) = \mathtt{HP}(\mathcal{S}') \cup \alpha(S')$ holds.

▶ **Lemma 5.8.** *Let $\mathcal{S}$ be a valid state, and let $\mathcal{S} = (R, \Delta, S) \to_r \mathcal{S}' = (R', \Delta', S')$ be a rule r application. Then, $\mathtt{HP}(\mathcal{S}) \cup \alpha(S) = \mathtt{HP}(\mathcal{S}') \cup \alpha(S')$ holds.*

Lemma 4.7 shows that our inference systems makes progress. Lemmas 5.8 and 4.7 imply that, for a given 1CU2r instance $\mathcal{I}$, when our inference system terminates, the obtained set of solutions $S \subset L_s$ satisfies that $\mathtt{HP}(\mathcal{I}) = \bigcup_{l \in S} \alpha(l)$.

Termination follows from the termination argument for the decision version. The termination argument, together with the correctness of our algorithm and the fact that all the rules can be checked for applicability and applied in polynomial time, gives us our main result stated in the following theorem.

▶ **Theorem 5.9.** *The 1CU2r problem can be solved in polynomial time. Moreover, a polynomial size representation of all solutions can be computed in polynomial time. This result holds also when a DAG is used for term representation.*

Finally, we informally argue that our representation satisfies the requirements stated at the beginning of Section 5. First, to extract just one solution, we can expand the definition of the concretization function α (Figure 2) in a depth-first traversal, without backtracking, until we get a concrete position. Second, to get the number m of all solutions, if there is a solution containing the expressions $\mathtt{exp}(p, k_1N + k_2)$ or $p.free$ then m is infinite, otherwise it is at most exponential and it can be efficiently computed using a dynamic programming scheme. Third, all solutions of $\mathcal{I}$ can be obtained in polynomial time w.r.t. m by a simple (depth-first) enumeration of the positions in the solution and expansion of expressions in `BaseCases`.

6 Easy Instances

Recall that our inference rules assume that easy instances of 1CU2r can be solved in polynomial time (Lemma 4.6); and that in fact, a representation for all solutions can also be efficiently computed. We establish those claims here, showing that Lemma 4.6 holds for all cases of Definition 4.5.

The special case where we have one equation that contains F on both sides was solved in polynomial time in a previous paper [8] (Theorem 5.20). That case corresponds, after

abstracting proper subterms of the form $F(r)$ by variables, to the case where $\mathcal{I}$ contains equations of the form $F(u_1) \doteq s, F(u_2) \doteq C[s]$ with $C \neq \bullet$. The key observation is that the hole position of F has to be a prefix or exponentiation of $\mathtt{hp}(C)$. In the problem considered in [8], the terms at the input were given explicitly. Since, we assume the DAG representation for terms, the result from [8] needs to be extended for our setting. However, the proof ideas are the same, and hence the proof is omitted here.

▶ **Lemma 6.1** (cyclic). *Let $\mathcal{I}$ be a 1-CU instance of the form $\mathcal{I} = \mathcal{I}' \cup \{F(u_1) \doteq s, F(u_2) \doteq C[s]\}$, where C is a non-empty context. Then, a polynomial time procedure* `SolveCyclic` *returns a complete representation of* $\mathtt{HP}(\mathcal{I})$ *of polynomial size.*

The expressions in the complete set of solutions of the previous lemma are of the forms $\mathtt{exp}(p, e)$ and $\mathtt{exp}(p, k_1N + k_2)$, where p is a position, e, k_1, k_2 are natural numbers bounded by $||\mathcal{I}||^2$, and N is an integer variable. The expressions $\mathtt{exp}(p, e)$ and $\mathtt{exp}(p, k_1N + k_2)$ stand for the result of raising p to e and $k_1N + k_2$, respectively.

Note that the fact that Lemma 4.6 holds for case (a) of Definition 4.5 follows from Lemma 6.1. The fact that Lemma 4.6 also holds for case (b) of Definition 4.5 follows from the following lemma.

▶ **Lemma 6.2** (Clash). *Let $\mathcal{I}$ be a 1CU instance of the form $\mathcal{I} = \mathcal{I}' \cup \{F(u_1) \doteq f(s_1, \ldots, s_m), F(u_2) \doteq g(t_1, \ldots, t_{m'})\}$, with $f \neq g$. Then, $\mathcal{I}$ can be solved by a polynomial time procedure* `SolveClash` *and* $\mathtt{HP}(\mathcal{I})$ *is either empty or* $\{\lambda\}$.

Proof. The fact that every solution σ must satisfy $F\sigma = \bullet$ directly follows from $f \neq g$. Hence, this particular case reduces to the first-order unification problem $\mathcal{I}' = \mathcal{I}\{F \to \bullet\}$, which can be solved in polynomial time w.r.t. $||\mathcal{I}||$. ◀

Let us now argue that Lemma 4.6 holds for cases (c) and (d) in Definition 4.5. Note that such cases reduce to solving one equation by unifying left hand-sizes and replacing first-order variables by their left hand-sides, respectively. Hence, these two cases follow from the following Lemma. Its proof is included in [9] and hence omitted here.

▶ **Lemma 6.3** (1-eqn). *Let $\mathcal{I}$ be a 1CU2r instance consisting of the single equation $F(s) \doteq t$. Then, a complete representation of* $\mathtt{HP}(\mathcal{I})$ *of polynomial size can be computed in polynomial time.*

The expressions in the complete set of solutions in the previous lemma are of the forms $\mathtt{hps}(u, v)$ and $p.free$ for terms u, v and a position p of polynomial size. These expressions stand for the (possibly exponential and infinite) sets of positions $\{q \in \mathtt{pos}(v) \mid v|_q = u\}$ and $\{p.p' \mid p' \in \mathtt{pos}(\mathcal{F})\}$, respectively.

Finally, we just need to show that Lemma 4.6 also holds for case (e) of Definition 4.5 to complete its proof. Note that such case reduces to solving an instance of the form $\{F(r) \doteq s, F(C[s]) \doteq t\}$. Before we give a polynomial time algorithm for that case, we first prove a particular case of 1CU: solving a single equation of the form $F(C[F(s)]) = t$. Equations of this form have the nice property that the hole position of any context that is a solution for F can not be an extension of a *nonlinear* positions in t. A position p is *nonlinear* in t if there exists another position $q \neq p$ such that $t|_p = t|_q$. We also call $t|_p$ a nonlinear subterm of t. Note that there are only a (linear number of) linear positions in a term. In contrast, there can be exponentially many nonlinear positions in a term. Since the following lemma is already proved in [9], we only state it here.

▶ **Lemma 6.4.** *Let $\mathcal{I}$ be a 1CU instance consisting of one single equation of the form $F(C[F(s)]) \doteq t$ such that F does not occur in t (but F can occur in C). Let $P = \{p \in \mathtt{pos}(t) \mid t|_p = v$ and v is a non-linear subterm of $t\}$. Then, $\forall p \in P : \mathtt{hp}(F\sigma) \neq p$, for every solution σ of $F(C[F(s)]) \doteq t$.*

It follows from the previous Lemma that for the equation $F(C[F(s)]) \doteq t$, we only need to test the (small number of) linear positions as possible hole positions. This implies that, as stated in the following lemma, we can enumerate a *complete* set of unifiers: a set of unifiers is *complete* if any other unifier (for the problem) is an instance of some unifier in the set. Here, a unifier is allowed to instantiate a context variable F in terms of a new context variable F'.

▶ **Lemma 6.5.** *Let $\mathcal{I}$ be a 1CU instance consisting of one single equation of the form $F(C[F(s)]) \doteq t$ such that F does not occur in t. Then, a complete set of unifiers U of $\mathcal{I}$ of polynomial size can be computed in polynomial time. Any substitution σ in U satisfies one of the two conditions below:*

1. *Either $F\sigma = t[\bullet]_p$, with $p \in \mathtt{pos}(t)$,*
2. *Or $\sigma = \{F \mapsto t[F'(\bullet)]_q,\ x \mapsto F'(C[t[F'(s)]_q])\}$, where x does not occur in $F(C[F(s)])$, $t|_q = x$, and F' is a new context variable different from F.*

Proof. We distinguish two cases. First consider solutions σ satisfying that $\mathtt{hp}(F\sigma) \in \mathtt{pos}(t)$. By Lemma 6.4, for each of such solutions, $\mathtt{hp}(F\sigma)$ must be in the set $Q = \{p \in \mathtt{pos}(t) \mid t|_p = v$ and v is a linear subterm of $t\}$. Note that $\{F \to t[\bullet]_p\} \leq \sigma$. Since $|Q|$ is polynomial w.r.t. $|t|$ even in the DAG representation, then U has a polynomial number of solutions of this form. Now consider solutions σ such that $\mathtt{hp}(F\sigma) \notin \mathtt{pos}(t)$. Let $p = p_1.p_2$, where p_1 is the longest prefix of p defined in t. Note that $t|_{p_1}$ must be a variable x linear in t by Lemma 6.4. Moreover, since σ satisfies $x\sigma|_{p_2} = C[F(s)]\sigma$, x does not occur in $C[F(s)]$. Hence, σ is of the form $\{F \mapsto t[D]_{p_1}, x \mapsto DC[t[D(s)]_{p_1}]\}$, for an arbitrary context D such that $\mathtt{hp}(D) = p_2$. Hence, all solutions σ such that $\mathtt{hp}(F\sigma) = q.q'$ and $q.q' \notin \mathtt{pos}(t)$, with $q \in Q$, can be represented by a substitution $\theta = \{F \mapsto t[[F'(\bullet)]]_q, x \mapsto F'(C[t[F'(s)]_q])\}$, where $t|_q = x$, F' is a new context variable different from F, since $\theta \leq \sigma$ holds. ◀

Finally, the fact that Lemma 4.6 holds for case (e) in Definition 4.5 follows from the following lemma, which completes the proof of Lemma 4.6. It is easy to see that the set of solutions of the next lemma can be represented using expressions $\mathtt{hps}(u, v)$ and $p.free$ for terms u, v and a position p of polynomial size, as in Lemma 6.3. Hence, Lemma 4.6 follows from Lemmas 6.1, 6.2, 6.3, and 6.6. Moreover a representation for all solutions consisting of expressions in `BaseCases` can always be computed for all easy instances.

▶ **Lemma 6.6.** *Consider a 1CU2r instance of the form $\mathcal{I} = \{F(r) \doteq s, F(C[s]) \doteq t\}$ Then, a complete representation of $\mathtt{HP}(\mathcal{I})$ of polynomial size can be computed in polynomial time.*

Proof. First note that if either s or t is a constant, the lemma is straightforward. Also, we can whether λ is in $\mathtt{HP}(\mathcal{I})$ by solving the first-order unification problem resulting from applying the substitution $F \mapsto \bullet$. Hence, we can assume that s and t are both not constants and $F\sigma \neq \bullet$, for every solution σ. By Lemma 6.5 we can compute, in polynomial time, a complete set of unifiers $U = \theta_1, \ldots, \theta_k$ of the single equation $F(C[F(r)]) \doteq t$ of polynomial size. Moreover, every substitution $\theta \in U$ satisfies one of the two conditions below:

1. $F\theta = t[\bullet]_p$, with $p \in \mathtt{pos}(t)$, or
2. $\theta = \{F \mapsto t[F'(\bullet)]_q, x \mapsto F'(C[t[F'(r)]_q])\}$, where x does not occur in $F(C[F(r)])$, $t|_q = x$, and F' is a new context variable different from F.

Hence, to obtain a polynomial time algorithm, it is enough to check if some substitution in U can be extended to solve also the equation $F(r) \doteq s$, and thus $\mathcal{I}$. Consider the two cases of a substitution $\theta \in U$.

1. If θ is such that $F\theta = t[\bullet]_p$, then the check can clearly be done in polynomial time, since $F(r)\theta \doteq s\theta$ is a first-order unification instance of polynomial size thanks to the DAG representation.
2. Otherwise, θ is of the form $\{F \mapsto t[F'(\bullet)]_q, x \mapsto F'(C[t[F'(r)]_q])\}$, where $t_q = x$, we distinguish cases depending on whether x occurs in s, and if so, where.
   - **(i)** First assume that x occurs in s at a position p such that either $p < q$ or $p > q$. Since we are looking for solution σ where $\mathtt{hp}(F\sigma) \geq q$, these cases can be rewritten into a form that is covered by Lemma 6.1. Note that since $p \neq q$, C in Lemma 6.1 is nonempty.
   - **(ii)**) Assume that, for some p, $s|_p = x$ and p is disjoint with q. In this case, every solution σ that is an extension of θ satisfies that $|F\sigma| > |F\sigma|_q\sigma| = |x\sigma| = |F(C[F(r)])\sigma| > |F\sigma|$, a contradiction. Hence we know that if x occurs in s at a position disjoint with q there are no solutions that are extensions of θ and thus there is no need to test θ.
   - **(iii)** Consider now the case where $s|_q = x$. In this case, every solution σ that is an extension of θ will satisfy $s\sigma = t\sigma$, and hence, also in this case, there is no need to test θ, since the equation $F(C[s]) \doteq t$, and our assumption that F is not $\bullet$ in any solution, imply $s\sigma \neq t\sigma$ for every solution σ.
   - **(iv)** Finally, consider the case where x does not occur in s. Note that if r contains x then $\mathcal{I}$ has no solution that is an extension of θ, again because of F is not $\bullet$ in any solution. Thus, neither $r\theta$ nor $s\theta$ contain F', $F(r)\theta \doteq s\theta$ reduces to a single equation $F'(r') \doteq s'$ after applying a few first-order decomposition steps, and the lemma follows from Lemma 6.3 in this case. ◀

7 Conclusion

We have shown that the subcase of the one context unification problem of two equations $F(r_1) = s_1, F(r_2) = s_2$ can be solved in polynomial time, including a polynomial size representation of all unifiers. Our procedure led to polynomial time algorithm for several subclasses of the general 1CU problem [9]. We leave it to future work to extend this work to the unrestricted one context unification problem, by extending the techniques presented in this paper and [9]. Another possible future line of research is to investigate the algorithmic properties of fault tolerant unification with other bounds on the number of faults, and whether our algorithm can be extended to obtain a polynomial time algorithm also for the case where STGs are used for term representation.

References

1 F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, New York, NY, USA, 1998.

2 F. Baader and W. Snyder. Unification theory. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 445–532. Elsevier and MIT Press, 2001.

3 H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree Automata Techniques and Applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2007. release October, 12th 2007.

4 C. Creus, A. Gascón, and G. Godoy. One-context Unification with STG-Compressed Terms is in NP. In *RTA*, pages 149–164, 2012.

5 S. Schulze Frielinghaus and H. Seidl M. Petter. Inter-procedural Two-variable herbrand Equalities. In *Proc. Symp. Programming, ESOP*, LNCS 9032, pages 457–482, 2015.

6 A. Gascón, G. Godoy, and M. Schmidt-Schauß. Context Matching for Compressed Terms. In *LICS*, pages 93–102, 2008.

7 A. Gascón, G. Godoy, and M. Schmidt-Schauß. Unification and Matching on Compressed Terms. *ACM Trans. Comput. Log.*, 12(4):26, 2011.

8 A. Gascón, G. Godoy, M. Schmidt-Schauß, and A. Tiwari. Context Unification with One Context Variable. *J. Symb. Comput.*, 45(2):173–193, 2010.

9 A. Gascón, M. Schmidt-Schauß, and A. Tiwari. One Context Unification Problems Solvable in Polynomial Time. In *LICS*, 2015. To appear.

10 W. D. Goldfarb. The Undecidability of the Second-Order Unification Problem. *Theor. Comput. Sci.*, 13:225–230, 1981.

11 S. Gulwani and A. Tiwari. Computing procedure summaries for interprocedural analysis. In *ESOP*, pages 253–267, 2007.

12 A. Jeż. Context unification is in PSPACE. In *ICALP*, pages 244–255, 2014.

13 J. Levy, M. Schmidt-Schauß, and M. Villaret. Monadic Second-Order Unification is NP-Complete. In *RTA*, pages 55–69, 2004.

14 J. Levy, M. Schmidt-Schauß, and M. Villaret. Bounded Second-Order Unification is NP-complete. In *RTA*, pages 400–414, 2006.

15 J. Levy, M. Schmidt-Schauß, and M. Villaret. Stratified Context Unification is NP-complete. In *IJCAR*, pages 82–96, 2006.

16 J. Levy, M. Schmidt-Schauß, and M. Villaret. The Complexity of Monadic Second-Order Unification. *SIAM J. Comput.*, 38(3):1113–1140, 2008.

17 J. Levy, M. Schmidt-Schauß, and M. Villaret. On the complexity of Bounded Second-Order Unification and Stratified Context Unification. *Logic Journal (IGPL)*, 19(6):763–789, 2011.

18 M. Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012.

19 M. S. Paterson and M. N. Wegman. Linear unification. *J. of Computer and System Sciences*, 16:158–167, 1978.

20 M. Schmidt-Schauß and K. U. Schulz. Solvability of Context Equations with Two Context Variables is Decidable. *J. Symb. Comput.*, 33(1):77–122, 2002.